Corpus transformations

Praktikum Verarbeitung natürlicher Sprache

Richard Mörbitz

14.06.2021

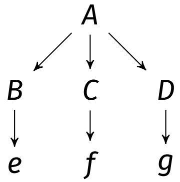
Praktical problems in parsing

Our basic CFG parser has some issues:

- Only binary and unary rules can be processed, but the training corpus is *n*-ary

$$A \rightarrow BC$$
$$A \rightarrow a$$

vs.



- Training corpus does not contain all English words

`He is a brilliant deipnosophist .` $\rightsquigarrow$ *No parse!*

- Sparse data problem: rare words in the training corpus may get atypical probabilities

Outline

Straight-forward binarization

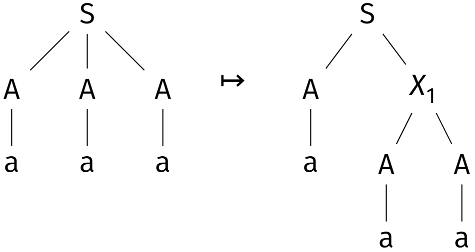
Binarizing the training corpus vs. binarizing the grammar

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

Straight-forward binarization

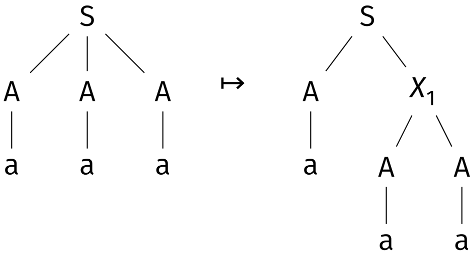
Binarizing the training corpus vs. binarizing the grammar



(1)

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



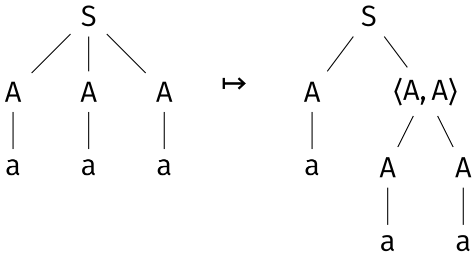
(1)

Problems:

1. Each occurence of two consecutive nonterminals yields a new nonterminal X_i

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



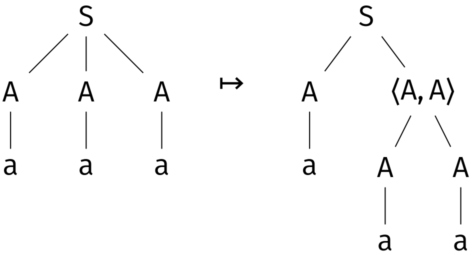
(2)

Problems:

1. Each occurence of two consecutive nonterminals yields a new nonterminal X_i

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



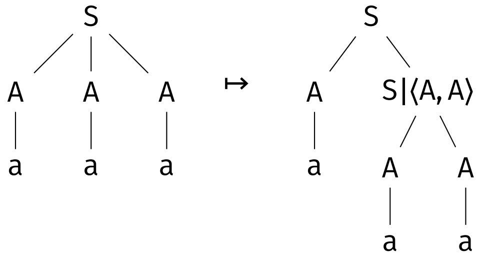
(2)

Problems:

1. Each occurence of two consecutive nonterminals yields a new nonterminal X_i
2. No information about context is preserved

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



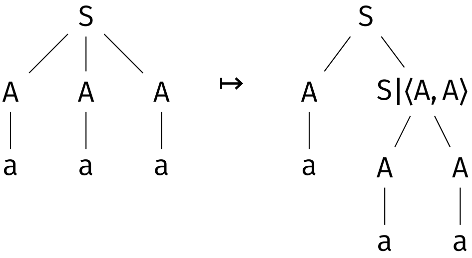
(3)

Problems:

1. Each occurence of two consecutive nonterminals yields a new nonterminal X_i
2. No information about context is preserved

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



(3)

Problems:

1. Each occurence of two consecutive nonterminals yields a new nonterminal X_i
2. No information about context is preserved
3. Number of right-hand side nonterminals is fixed

Vertical and horizontal Markovization [1]

Previous slide Markovization with $v = 1$ and „$h = \infty$"

In general store $v \in \mathbb{N}_+$ nonterminals towards the root and $h \in \mathbb{N}_+$ nonterminals to the left

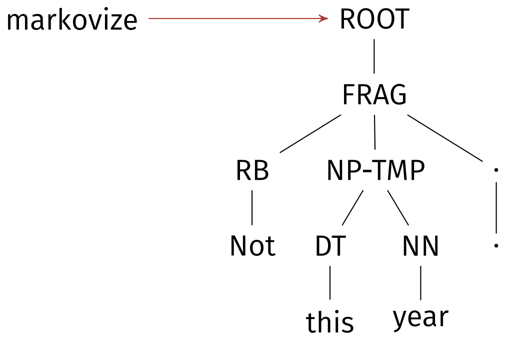
function MARKOVIZE($t = \sigma(t_1, \ldots, t_k)$)
    if t is preterminal **then**
        return t
    else if $k \leq 2$ **then**
        return ADD_PARENTS(σ)(MARKOVIZE(t_1), … , MARKOVIZE(t_k))
    else
        $\sigma' \leftarrow$ ORIGINALLABEL(σ)|⟨label of t_2, …, label of t_{h+1}⟩
        return ADD_PARENTS(σ)(MARKOVIZE(t_1), MARKOVIZE($\sigma'(t_2, \ldots, t_k)$))

Note:

- ADD_PARENTS(σ) = σ^$\langle l_1, \ldots, l_{v-1} \rangle$, where the l_i are the labels of the ancestors of σ which occur in the original tree
- If $v = 1$ or there are no parents, leave out ^⟨⟩

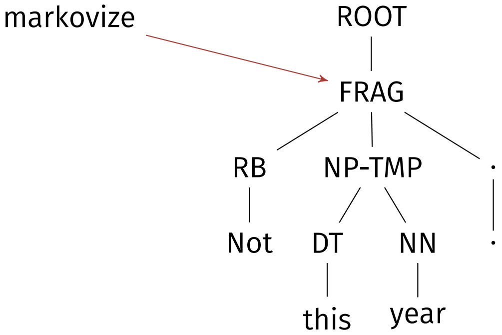
Markovization – example

here: $v = 3$, $h = \infty$



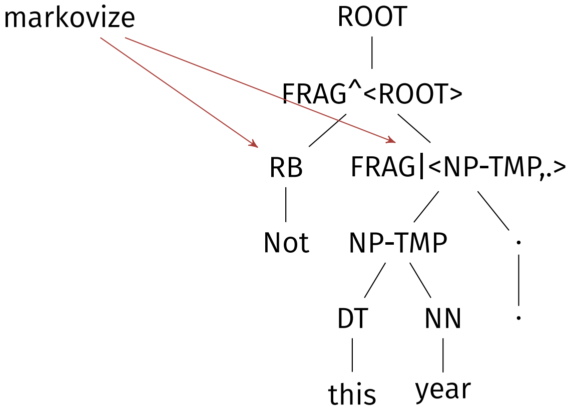
Markovization – example

here: $v = 3$, $h = \infty$



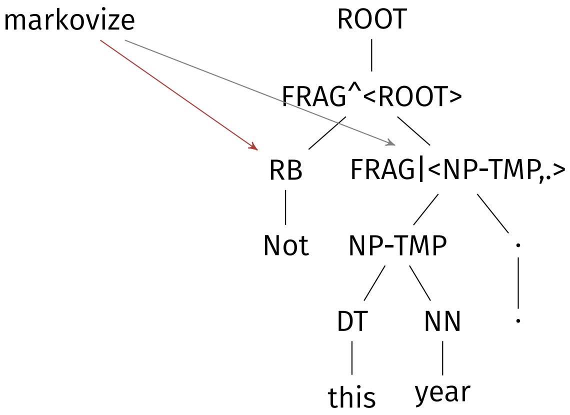
Markovization – example

here: $v = 3$, $h = \infty$



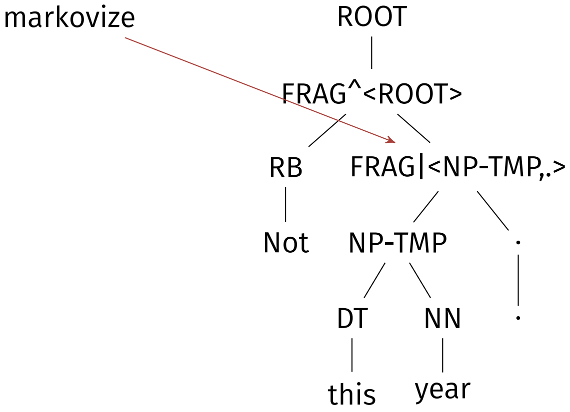
Markovization – example

here: $v = 3$, $h = \infty$



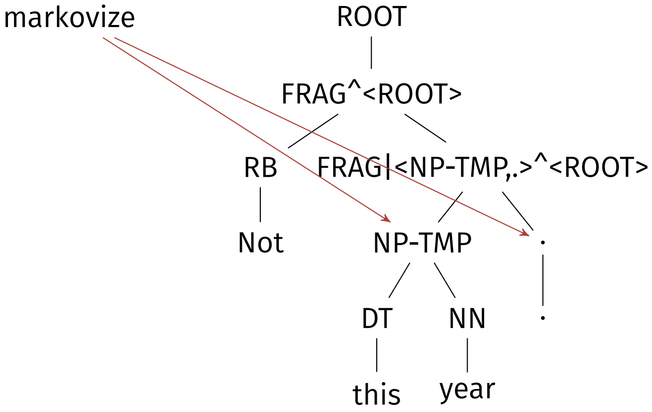
Markovization – example

here: $v = 3$, $h = \infty$



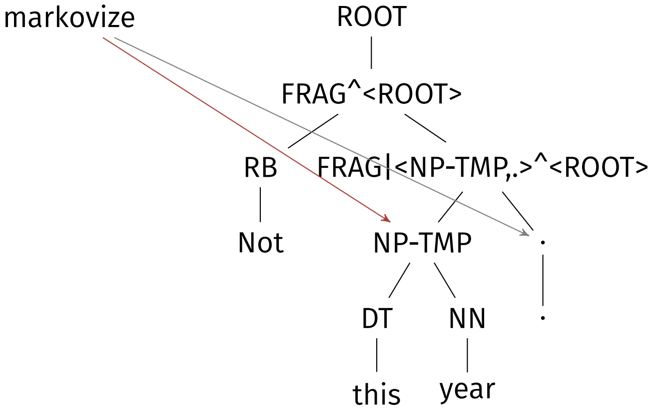
Markovization – example

here: $v = 3$, $h = \infty$



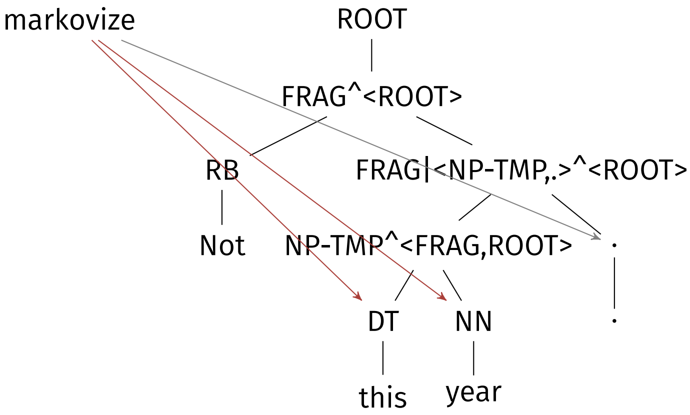
Markovization – example

here: $v = 3$, $h = \infty$



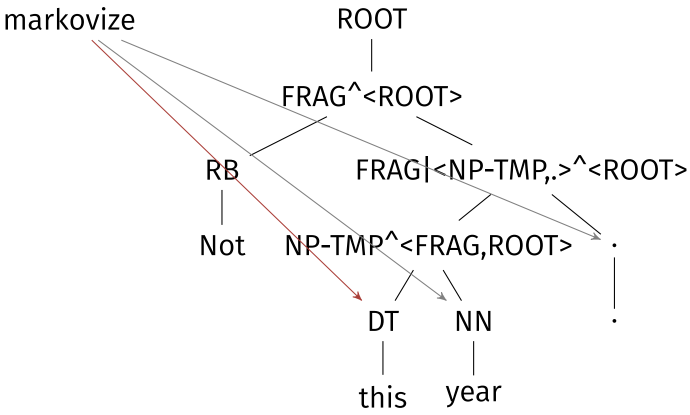
Markovization – example

here: $v = 3$, $h = \infty$



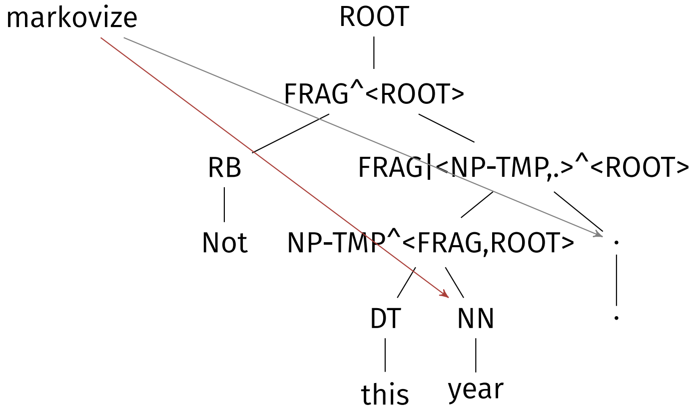
Markovization – example

here: $v = 3$, $h = \infty$



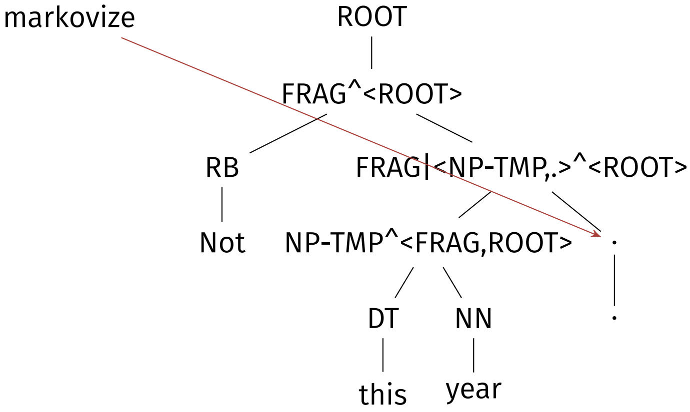
Markovization – example

here: $v = 3$, $h = \infty$



Markovization – example

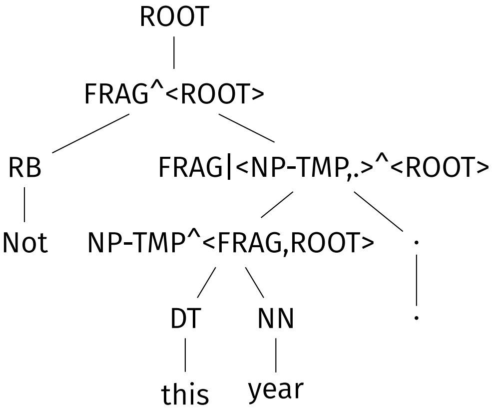
here: $v = 3$, $h = \infty$



Markovization – example

here: $v = 3$, $h = \infty$

markovize



After parsing: debinarization

Why?

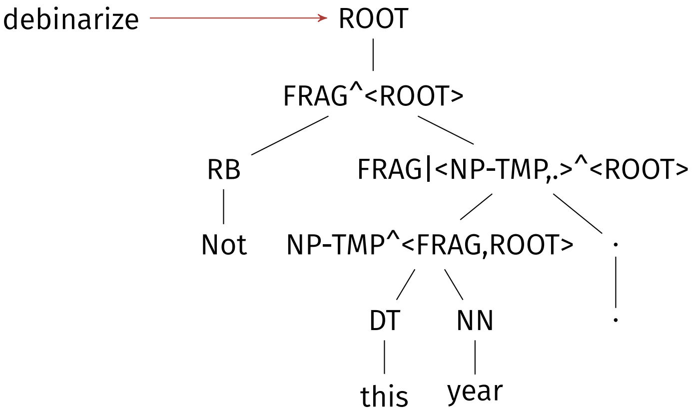
- Debinarized parse trees are linguistically relevant
- Comparison with (debinarized) gold corpus

function DEBINARIZE($t = \sigma(t_1, \ldots, t_k)$)
    if ROOT(t_k) is Markovization node with children t'_1, t'_2 **then**
        return DEBINARIZE($\sigma(t_1, \ldots, t_{k-1}, t'_1, t'_2)$)
    else
        return $\sigma\big($DEBINARIZE(t_1), …, DEBINARIZE(t_k)$\big)$

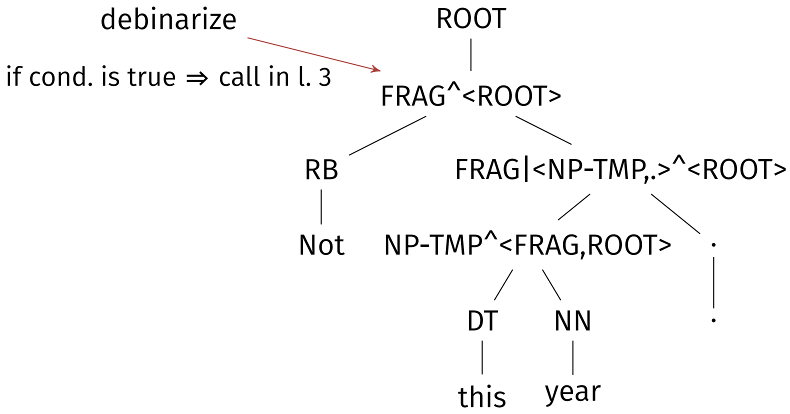
Also: remove ancestor annotation from each node (not shown)!

In material: both binarized and (debinarized) gold corpus

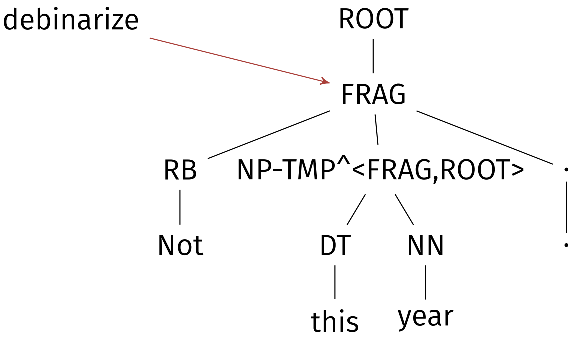
Debinarization – example



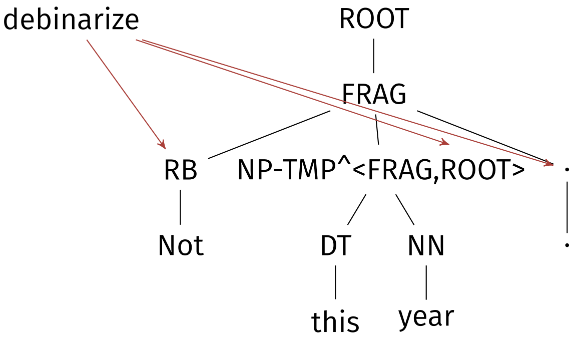
Debinarization – example



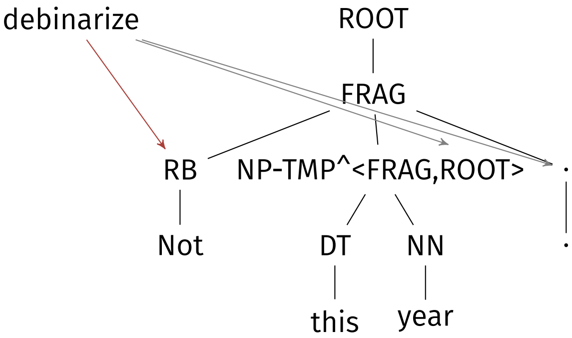
Debinarization – example



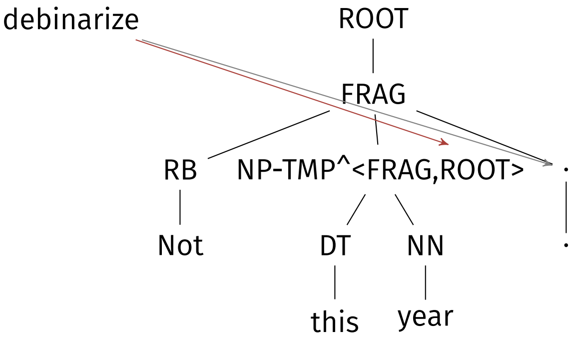
Debinarization – example



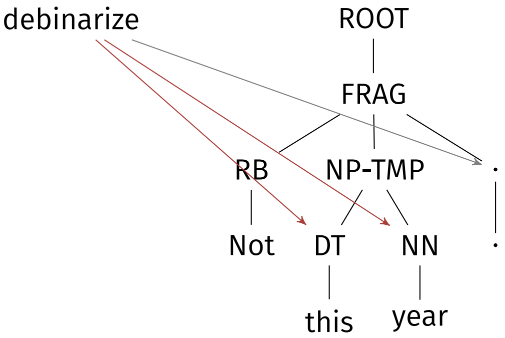
Debinarization – example



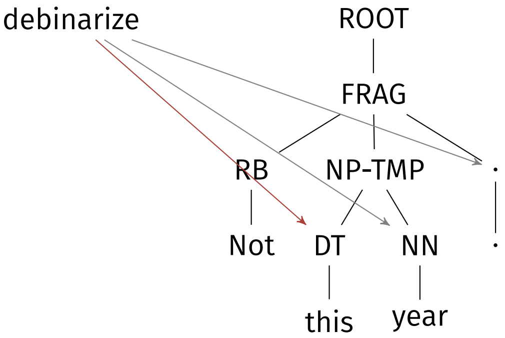
Debinarization – example



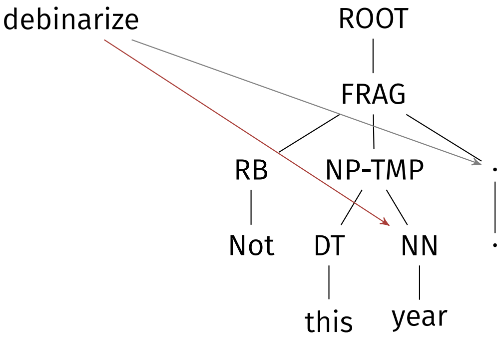
Debinarization – example



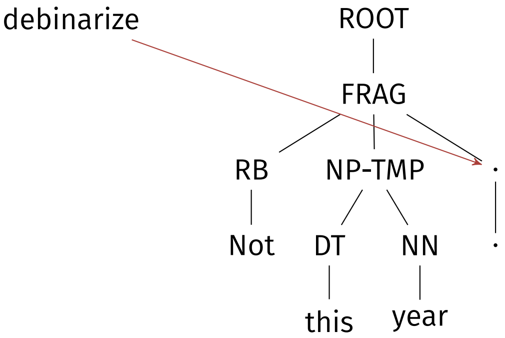
Debinarization – example



Debinarization – example

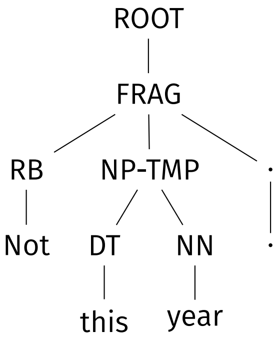


Debinarization – example



Debinarization – example

debinarize



Outline

Basic unking

Idea:

1. Replace words that are *unknown* to the parser by an „unknown token“ (here: `UNK`)
2. Assign some probability mass to rules that generate `UNK`

Basic unking

Idea:

1. Replace words that are *unknown* to the parser by an „unknown token“ (here: `UNK`)
2. Assign some probability mass to rules that generate `UNK`

Implementation:

1. Reflection in the parser: before parsing w (where Σ is the set of terminals)

   $wmap \leftarrow (w_i \mid i \in \{1, \ldots, |w|\})$
   for $i = 1, \ldots, |w|$ **do**
       if $w_i \notin \Sigma$ **then**
           $w_i \leftarrow \texttt{UNK}$

   and after parsing w as t, restore original words

   for $i = 1, \ldots, |w|$ **do**
       $\textsc{leaves}(t)[i] \leftarrow wmap[i]$

Basic unking (2)

2. Reflection in the grammar: replace rare words in the training corpus by `UNK`

Require: tree corpus *corpus* with terminal alphabet Σ, *threshold* $\in \mathbb{N}_+$
Ensure: every word occuring $\leq$ *threshold* times in *corpus* is replaced by `UNK`

$wordcount \leftarrow (0 \mid i \in \Sigma)$
for $t \in corpus$ **do**
    for $i = 1, \ldots, |\text{LEAVES}(t)|$ **do**
        $wordcount[\text{LEAVES}(t)[i]] \leftarrow wordcount[\text{LEAVES}(t)[i]] + 1$
for $t \in$ corpus **do**
    for $i = 1, \ldots, |\text{LEAVES}(t)|$ **do**
        if $wordcount[\text{LEAVES}(t)[i]] \leq threshold$ **then**
            $\text{LEAVES}(t)[i] \leftarrow$ `UNK`

The grammar is induced from the modified corpus!

Refinement

Observation: each unknown word is assigned the same probability

- bad language model: certain words are more likely to occur
- may worsen the parsing of sentences with rare words

Solution: categorize unknown words based on their *signature* [1]

$$w_i \leftarrow \mathtt{UNK} \quad \rightsquigarrow \quad w_i \leftarrow \textsc{getSignature}(w_i, i)$$
$$\textsc{leaves}(t)[i] \leftarrow \mathtt{UNK} \quad \rightsquigarrow \quad \textsc{leaves}(t)[i] \leftarrow \textsc{getSignature}(w_i, i)$$

- Some signatures can be found in the source code of the Berkely parser
- Next slide: „unknownLevel = 4“
- Heuristics, prone to overfitting
- Be creative, but don't overdo it!

```
function GETSIGNATURE(word, i)
    if |word| = 0 then return UNK
    letterSuffix ← ISUPPER(word_1) ∧ NONE(ISLOWER, word)  ⇒  -AC
                   ISUPPER(word_1) ∧ i = 1                ⇒  -SC
                   ISUPPER(word_1)                        ⇒  -C
                   ANY(ISLOWER, word)                     ⇒  -L
                   ANY(ISLETTER, word)                    ⇒  -U
                   otherwise                              ⇒  -S
    numberSuffix ← ALL(ISDIGIT, word)  ⇒  -N
                   ANY(ISDIGIT, word)  ⇒  -n
                   otherwise           ⇒  ε
    dashSuffix ← ANY((= '-'), word)  ⇒  -H
                 otherwise           ⇒  ε
    periodSuffix ← ANY((= '.'), word)  ⇒  -P
                   otherwise           ⇒  ε
    commaSuffix ← ANY((= ','), word)  ⇒  -C
                  otherwise           ⇒  ε
    wordSuffix ← |word| > 3 ∧ ISLETTER(word_|word|)  ⇒  TOLOWER(word_|word|)
                 otherwise                           ⇒  ε
    return UNK · letterSuffix · numberSuffix · dashSuffix · periodSuffix · commaSuffix · wordSuffix
```

What to do?

Until 14.07.2020, 23:59,

- implement debinarization (3a)
- implement trivial unking for corpus and adapt parser (3b)
- implement 3 of
  - Markovization
  - smoothing
  - pruning
  - *n*-best parsing
  - heuristic search

  } optimizations, next week

All tasks' solutions are one submission!
You may send in your solutions earlier for feedback.

References I

[1] Dan Klein und Christopher D Manning. „Accurate unlexicalized parsing“. Association for Computational Linguistics. 2003.